Research Paper | Education | E-ISSN No : 2454-9916 | Volume : 9 | Issue : 8 | August 2023

IMPLEMENTATION OF WALLACE TREE MULTIPLIER USING XILINX WITH VERILOG

G. Appala Naidu

Assistant Professor, ECE Department, JNTUGV-CEV, Vizianagaram

ABSTRACT

Background: This paper presents a 32-bit high-speed Wallace tree multiplier designed using Verilog HDL. The multiplier achieves efficient multiplication by reducing partial products and employs carry-save and ripple carry adders. Practical implementation on FPGA demonstrates its superiority over existing methods in terms of computation time and efficiency. The proposed design addresses the area and speed disadvantages of traditional Wallace tree multipliers, offering a promising solution for high-speed multiplication applications. In this paper a 32-bit high speed Wallace tree multiplier is designed using Verilog and 4-bit multiplication is implemented on Spartan-6 LX45 FPGA and. The entire design of 4-bit multiplication is coded in Verilog HDL, simulated with Modelsim and synthesized using Xilinx FPGA device and the similar algorithm is used for computing 32-bit multiplication. The proposed multiplier is much efficient than the existing methods.

KEYWORDS: Wallace Tree, Multiplexer, FPGA.

INTRODUCTION

Multiplication is the basic arithmetic operation and is widely used everywhere during computation. In digital signal processing, most of the arithmetic operations require the use of multiplications. The performance of three-dimensional computer graphics mostly depends on the performance of multiplications. Therefore, there has been much work on advanced multiplication algorithms and design also. Critical factors in the design of multipliers are chip area and speed of multiplication. There is highly demand of high-speed multiplications and require less hardware. The performance of multiplier is affected by the multiplication strategy and type of the multiplier used.

Multiplication involves two basic operations, the generation of the partial products and their sum, performed using two kinds of multiplication algorithms, serial and parallel. Serial multiplication algorithms use sequential circuits with feedbacks: inner products are sequentially produced and computed. Parallel multiplication algorithms often use combinational circuits and do not contain feedback structures. The common multiplication method is “add and shift” algorithm determines the performance of the multiplier. To reduce the number of partial products to be added, Modified Booth algorithm is one of the most popular algorithms. To achieve speed improvements “Wallace Tree algorithm” can be used to reduce the number of sequential adding stages.

TYPES OF MULTIPLIERS

The Multipliers are essential component in computation. It can be classified as Array Multiplier, Dadda Multiplier, Booth Multiplier, Modified Booth Multiplier, Baugh-Wooley Multiplier. Each section can explained in detailed below.

a. Array Multiplier:

Array multiplier is a regular structure multiplier; it uses the standard add and shift algorithm. The partial product is generated by multiplying the multiplicand with each bit of the multiplier. Array multiplier consumes more power and optimum number of components required. It requires larger number of gates, and therefore the area also increases. So, the array multiplier performs poor in terms of area. It is a fast multiplier but hardware complexity increases due to the large number of gates.

b. Dadda Multiplier:

The Dadda multiplier is a hardware binary multiplier design invented by computer scientist Luigi Dadda in 1965. The Dadda multiplier is a binary hardware multiplier that efficiently multiplies two numbers using a reduction tree of full and half adders. It aims to minimize gate usage and input/output delay. The reduction process follows specific rules based on a maximum-height sequence. This results in a slightly longer but more optimized final product, requiring larger adders.

c. Booth Multiplier:

This algorithm was devised by Andrew Donald Booth in 1950. It is extremely prevailing algorithm for signed number multiplication, which considers both positive and negative number uniformly. An algorithm uses 2’s complement presentation of signed binary number for multiplication. Each multiplier bit generates one multiple of the multiplicand that should be added to the partial product. In this case, the delay of the multiplier is mainly governed by number of additions to be performed. If there is way to reduce the number of additions, the performance will be better. It is the standard technique used in chip design and provides significant improvement over the long multiplication.

d. Modified Booth Multiplier:

Another improvement in the multiplier is by decreasing the number of individual partial products. To achieve high speed multiplication, algorithm using parallel counter like modified booth algorithm has been demonstrated and practiced. It reduces number of multiplicand and number of partial products. It groups three consecutive bits at a time, in that three-bit 2 bits on left hand side are present bit and third bit is the carry bit from previous pair of bits. The multiplier operates much faster than array multiplier for longer operands because it’s time to complete the operation is proportional to the log of the word length of operand. The main benefit of this multiplier is that if the successive bits in multiplicand are same, then addition can be skipped. The number of partial products is downed by a factor of half by using the technique of booth recoding. The grouping of bits and booth recording determine the number of partial products.

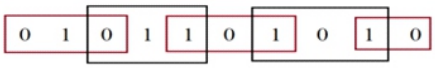

Fig 1.10: Grouping of bits from multiplier terms

Block Diagram:

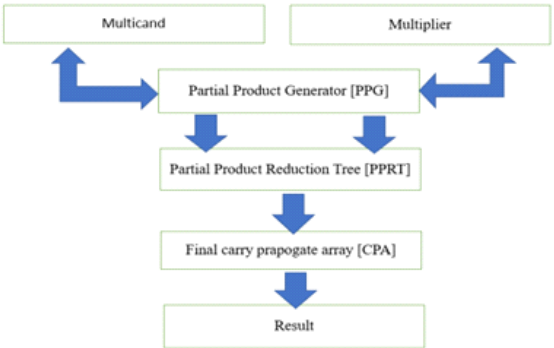

Fig 1.11: Block diagram of Modified Booth algorithm

The partial product generator [PPG] generates the partial product from the multiplicand and encoded multiplier. The partial product obtained at the output of PPG are then added by PPRT without carry propagation. Finally, the summed result obtained from PPRT stage are added using carry propagation array [CPA].

e. Baugh-Wooley Multiplier

In signed multiplication the length of the partial products and the number of partial products will be very high. So, an algorithm was introduced for signed multiplication called as Baugh-Wooley algorithm. The Baugh-Wooley multiplication is one amongst the cost-effective ways to handle the sign bits. This method has been developed so as to style regular multipliers, suited to 2's compliment numbers.

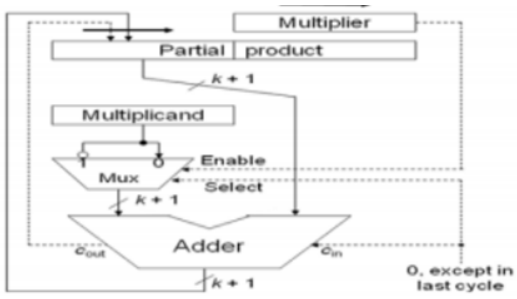

Fig 1.12: Hardware implementation of Baugh-Wooley Multiplier

Let two n-bit numbers, number (A) and number (B), A and B are often pictured as

$$A = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$$

$$B = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i$$

Where a_i and b_i are a unit the bits during A and B, severally and

The full precision product, P = A × B, is provided by the equation: The first two terms of above equation are positive and last two terms are negative. In order to calculate the product, instead of subtracting the last two terms, it is possible to add the opposite values. The above equation signifies the Baugh-Wooley algorithm for multiplication process in two's compliment form. Baugh-Wooley Multiplier provides a high speed, signed multiplication algorithm. It uses parallel products to complement multiplication and adjusts the partial products to maximize the regularity of multiplication array. When number is represented in two's complement form, sign of the number is embedded in Baugh-Wooley multiplier. This algorithm has the advantage that the sign of the partial product bits are always kept positive so that array addition techniques can be directly employed. In the two's complement multiplication, each partial product bit is the AND of a multiplier bit and a multiplicand bit, and the sign of the partial product bits are positive.

METHODOLOGY

A Wallace multiplier is a hardware implementation of a binary multiplier, a digital circuit that multiplies two integers. It uses a selection of full and half adders (the Wallace tree or Wallace reduction) to sum partial products in stages until two numbers are left. Wallace multipliers reduce as much as possible on each layer, whereas Dadda multipliers try to minimize the required number of gates by postponing the reduction to the upper layers.

The Wallace tree has three steps:

1. Multiply each bit of one of the arguments, by each bit of the other.
2. Reduce the number of partial products to two by layers of full and half adders.
3. Group the wires in two numbers and add them with a conventional adder.

Compared to naively adding partial products with regular adders, the benefit of the Wallace tree is its faster speed. It O(log n) has reduction layers, but each layer has only O(1) propagation delay. A naïve addition of partial products would need $O(\log_2 n)$ time. As making the partial products is and the final addition is , the total multiplication is O(log n), not much slower than addition. From a complexity theoretic perspective, the Wallace tree algorithm puts multiplication in the class NC^1. The downside of the Wallace tree, compared to naive addition of partial products, is its much higher gate count.

The first step, as said above, is to multiply each bit of one number by each bit of the other, which is accomplished as a simple AND gate, resulting in bits; the partial product of bits by has weight.

In the second step, the resulting bits are reduced to two numbers; this is accomplished as follows: As long as there are three or more wires with the same weight add a following layer:

1. Take any three wires with the same weights and input them into a full adder. The result will be an output wire of the same weight and an output wire with a higher weight for each three input wires.
2. If there are two wires of the same weight left, input them into a half adder.
3. If there is just one wire left, connect it to the next layer.

In the third and final step, the two resulting numbers are fed to an adder, obtaining the final product.

				a_3	a_2	a_1	a_0
				x	b_3	b_2	b_1	b_0
p_{70}	p_{60}	p_{50}	p_{40}	p_{30}	p_{20}	p_{10}	p_{00}
p_{61}	p_{51}	p_{41}	p_{31}	p_{21}	p_{11}	p_{01}	x
p_{52}	p_{42}	p_{32}	p_{22}	p_{12}	p_{02}	x	x
p_{43}	p_{33}	p_{23}	p_{13}	p_{03}	x	x	x
z_7	z_6	z_5	z_4	z_3	z_2	z_1	z_0

A – holds Multiplicand
B – holds Multiplier
p – Partial Product

The partial product generator generates partial products using a simple two-input AND gate that is fed to the Wallace tree adder. Multiple halves and a full adder that does additions in multiple levels and also considers carry generated by a previous level adder. The last level of the Wallace tree adder can be implemented ripple carry adder. However, to improve computation latency, a carry look-ahead adder can also be used.

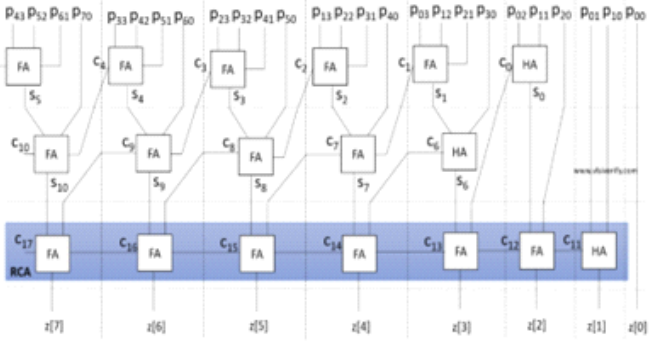

Fig 2.2: Block diagram of Wallace Tree Multiplier

SIMULATION RESULT

The simulation of 32 bit Wallace tree multiplier implemented using ModelSim SE 10.6d. The simulation is performed successfully after checking the errors using test bench to produce inputs for the code and in simulation we have considered the two different values of 'a' and 'b'. Consider a = 7Bh, b= Bh then product of a and b is 549h. If a = 5345h and b = 123h, then product of a and b is 5ea76fh. The above two examples are simulated in Modelsim by using the algorithm of 32 bit Wallace tree multiplier. The simulation results of those two examples are given below:

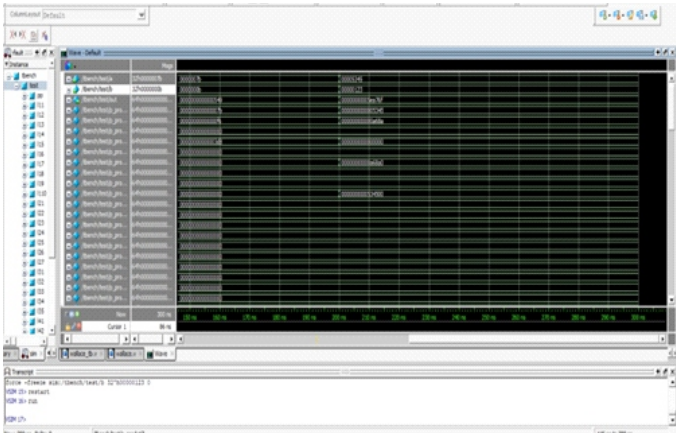

Fig 5.2: Simulation results using Modelsim

When the inputs as two 32 bit size numbers and the simulator uses the algorithm inside the code and produces a output of size 64 bit which is the multiplication of those two 32 size bits.

SYNTHESIS RESULTS

Synthesis of 32-bit Wallace tree multiplier is done by using Xilinx ISE Design Suite 14.7.Synthesis is simply a step of converting our code into circuit. For synthesis purpose we cannot require the test bench. The schematic diagram of high-Speed 32-bit Wallace tree multiplier is shown in fig 6.1 Schematic view of Wallace tree multiplier.

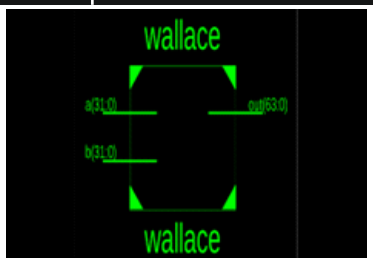

Fig 6.1: Schematic view of 32-bit Wallace tree multiplier

At the last we are using two 32-bit ripple carry adders in order to generate 64 bit product result. one single carry bit and the outputs are 32 bit sum and one carry bit and in place of ripple carry adders we can also use carry look ahead adder and the fig 6.5 represents the RTL view of 32 bit ripple carry adder.

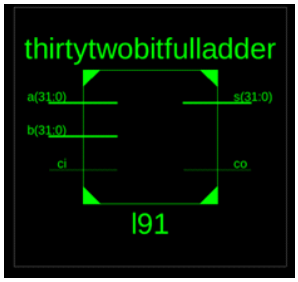

Fig 6.5: Schematic view of ripple carry adder.

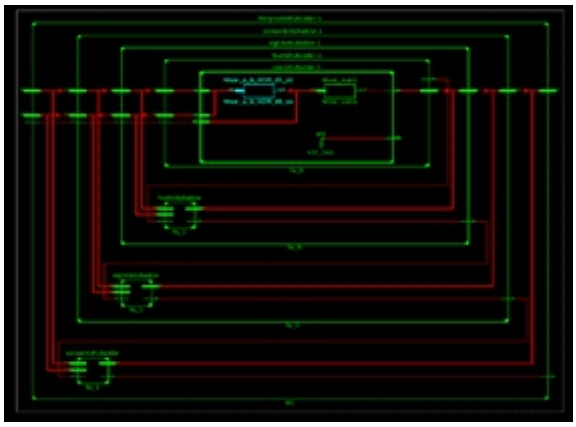

Fig 6.6: RTL view of ripple carry adder.

The overall RTL view of High speed 32-bitwallace tree multiplier which consists of all the above modules which are generation of partial products and many number of carry save adders and ripple carry adders is shown in the fig 6.6. The delay of 32 bit Wallace tree multiplier is 17.982ns which is very less delay compared to other multipliers. The area occupied by the 32 bit Wallace tree multiplier is number of slice LUTs are 2004.

SIMULATION RESULTS

Simulation plays an important role in the design of integrated circuits. Using simulation, a designer can determine both the functionality and the performance of a design before the expensive and time-consuming step of manufacture.

The simulation results is also performed by using Xilinx by using ISim and the example if a = 123, b = 11 then the product of 'a' and 'b' is 1353.

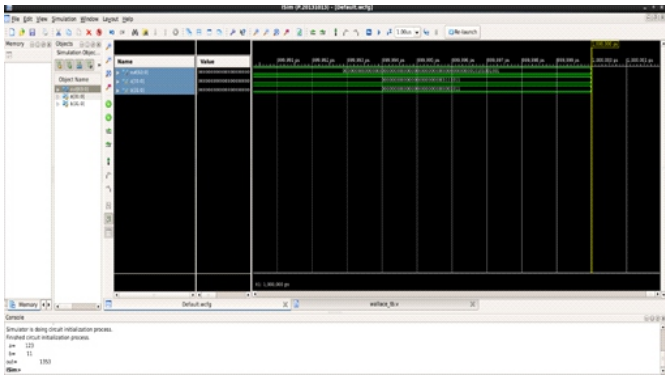

Fig 6.13: Simulation results using Xilinx.

```
This is a Full version of ISim.
Time resolution is 1 ps
# onerror resume
# wave add /
Time resolution is 1 ps
# onerror resume
# wave add /
# run 1000 ns
Simulator is doing circuit initialization process.
Finished circuit initialization process.
  a=       123
  b=        11
out=              1353
# run 1000 ns
```

Fig 6.14: Simulation result shown in console window.

4-bit Wallace Tree implementation on Spartan-6 Lx45

We have successfully implemented the hardware implementation 4 bit Wallace tree multiplier by dumping bit program onto Spartan-6 LX45 FPGA board.We have performed the hardware implementation for two different cases of A and B and their product(prod) is shown by leads on the FPGA board. Inputs for the multiplier is given to FPGA using 8 switches on the board, in which4 switches for one operand and another four switches for other operand.

Case 1: A = 1111
B = 1111
Product is 11100001

Fig 8.6: Output on FPGA for case 1

Case 2: A = 1011
B = 1011
Product is 01111001

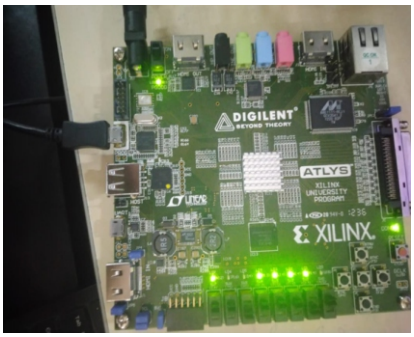

Fig 8.7: Output on FPGA for case 2

CONCLUSION

The Objective of this work Implementation of Wallace tree multiplier with xilinx using verilog is we have implemented the 32 bit wallace tree multiplier in Modelsim and Xilinx ISE Design Suite14.7 and we found the area of the 32-bit Wallace tree multiplier is 2004 slice LUTs and the delay is 17.982ns and the algorithem which we have used is modified one and 4 bit wallace tree multiplier is implemented on SPARTAN-6 LX45 development kit and.The entire design of 4-bit and 8-bit multiplication is coded in Verilog HDL ,simulated with Modelsim and synthesized using Xilinx FPGA device and the similar algorithm is used for computing 32-bit multiplication .In order to provide an efficient wallace tree multiplier we have compared the 8 bit normal wallace tree multiplier consists of regular full adders full adders with multiplexer based wallace multiplier and compressor based multiplier ..The proposed multiplier is much efficient than the existing multipliers and we have also overcome the disadvantage of normal wallace tree multiplier which requires more area and more delay. We also found that the area usage increased as the bit width of the multiplier increased, which is a common trade-off in digital circuit design.

Overall, our experiment demonstrates the importance of choosing an appropriate multiplier architecture for specific applications. The Wallace tree multiplier is a viable option for high-speed and area-efficient multiplication, and can be optimized by choosing the right design and bit width. Future research can explore other optimization techniques for the Wallace tree multiplier, such as the use of carry-save adders and parallel-prefix adders.

REFERENCES

1. S. Ravi Chandra Kishore and K.V. Ramana Rao "Implementation of carry-save adders in FPGA" IJEAT ISSN: 2249 – 8958, Volume-1, Issue-6, (2012), pp.27-30.
2. J.-L. Beuchat and J.-M. Muller, "Automatic generation of modular multipliers for

FPGA applications," IEEE Transactions on Computers, vol. 57, no. 12, (2008) pp. 1600–1613.
3. J. Detrey, F. de Dinechin, and X. Pujol, "Return of the hardware floating-point elementary function," in Proceedings of the 18th IEEE Symposium on Computer Arithmetic (Montpellier, France), Kornerup and Muller, Eds. Los Alamitos, CA: IEEE Computer Society Press, (2007), June 25-27.
4. H. Eberle, G. N., S. Shantz, V. Gupta, L. Rarick, and S. Sundaram, "A public-key cryptographic processor for RSA and ECC," in Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors, (2004), September 29-29.
5. H. R. Ismail, R.C., "High performance complex number multiplier using booth-wallace algorithm," in IEEE International Conference on Semiconductor Electronics ICSE, (2006), November 29- December 1.
6. K. Manochehri and S. Pourmozafari, "Modified radix-2 montgomery modular multiplication to make it faster and simpler," in IEEE International Conference on Information Technology: Coding and Computing, (2005), April 4-6.
7. Bhawna Singroul and Pallavee Jaiswal, "A review on performance Evaluation different digital multiplier in VLSI using VHDL", International journal of Engineering research & technology, vol-7, issue-5, (2018), pp.57-64.
8. Savita Nair, "A review paper on comparison of multiplier based on performance parameter", International journal of computer application, vol-2, (2014), pp 6-9.
9. Sumit Vaidya and Deepak Dandekar, "A review on delay performance comparison of multiplier in VLSI circuit design", International journal of computer network & communication, Vol 2, issue 4, (2010), pp 47-55.
10. Shweta S. Khobragade and Swapnil P. Kormore, "A review on low power VLSI design of Modified Booth multiplier", International journal of Engineering and Advanced Technology, vol-2, issue-5, (2015), pp 463-466.
11. K. N. Swamy and Jami Venkata Suman, "Design of Optimized Multiply Accumulate Unit Using EMBR Techniques for Low Power Applications", In Computational Intelligence in Data Mining, Springer, vol. 2, (2016), pp. 315-323.
12. Bhavya Lahari Gundapaneni and JRK kumar Dabbakutti, "A review on Booth Algorithm for the design of multiplier", International journal of innovative technology and Exploring Engineering, vol-8, issue-7, (2019), pp 1506-1509.